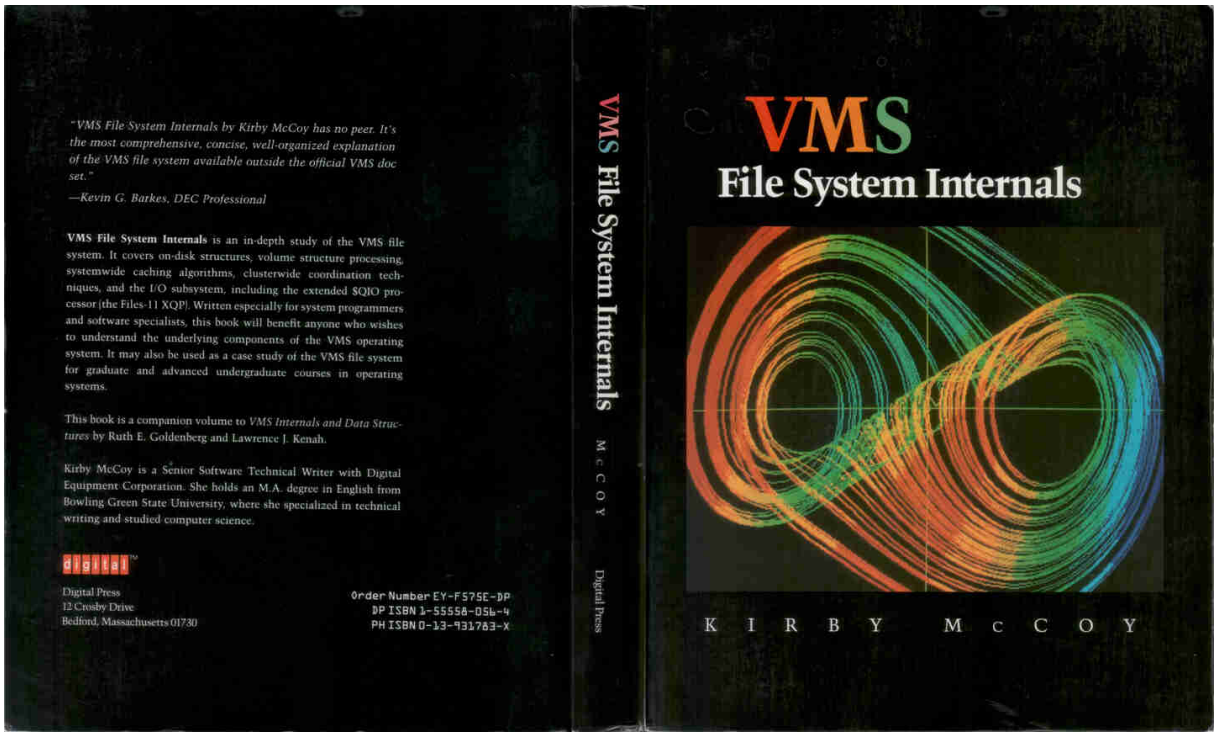
# RETROSPECTIVE REVIEW – VMS FILE SYSTEM INTERNALS

## 1. INTRODUCTION

Sometimes, one's fancy is caught by the notion of learning in some detail the architecture and implementation of a piece of software lying outside one's usual domain of expertise or area of activity. So with file management systems – an operating system component that is relied upon constantly, but whose innards few people get acquainted with, and even fewer get to design. Curiosity may lead one to consider not just current, cutting-edge software (which certainly brings the advantage of learning about a system one may well have to use professionally), but rather an old, even obsolete one (with the perspective of learning how certain problems were addressed when technology did not afford solutions commonly used nowadays).




VMS makes for an interesting case: it marks the transition from mainframes, with magnetic tapes as the main storage medium, to time-sharing minicomputers based on disk drives. Its heyday corresponds to a rapid evolution of hard disks in terms of technology, performance and capacity. Simultaneously, storage services offered by operating systems were increasingly abstracting away low-level hardware characteristics so that programmers no longer had to care about the peculiarities of blocks and cylinders to organize their files.

Of course, many software engineers probably read descriptions of the original Unix file system, but the VMS file system is much less known, despite VMS having

been dominant for a decade. Since both operating systems were developed at about the same time on the same machines, a comparison may be intriguing. Let us then see whether the book "*VMS File System Internals*" by Kirby McCoy enlightens us about what engineers at DEC wrought.

## 2. CONTENTS

This book deals with Files-11/ODS-2, the file management component of VAX/VMS machines from Digital Equipment Corporation (DEC).

After a general presentation of the system, a series of chapters considers each of its aspects in detail.

- Chapter 2 describes the persistent data structures defining the file system: boot block, home block, storage bitmap, file descriptors (covering attributes such as identity, space allocation table, access control information, creation date), directory hierarchy, etc.
- Chapter 3 explains how disks are formatted, mounted, and dismounted. VMS optimizes I/O by caching disk blocks.
- Chapter 4 shows how interrelated pools of disk blocks, distinguished by type (file headers, data, directory entries, allocation bitmap, quotas, etc) are structured and organized, and how buffers therein are allocated and freed.
- Chapter 5 covers the fundamental operations to access, create, modify, and delete files, as well as to manage directories, disk space and quotas. Emphasis is put on the integrity checks performed by each operation, and how "windows" serve to access file extent tables efficiently.
- Chapter 6 delves into lower-level I/O routines: how I/O requests are formatted, device drivers invoked, and results returned to applications.
- Chapter 7 examines file access serialization, including the various kinds of locks, locking levels, and the schemes relied upon to avoid deadlock.
- The final chapter explains how distributed locks synchronize accesses to files in a VAXCluster environment, how caches in participating computers are kept consistent, and how space allocation is handled.

Importantly, the book does not deal with the Record Management Services (RMS), the module used by VMS programmers to manipulate files as persistent high-level data structures such as indexed-sequential storage.

## 3. PRESENTATION AND STYLE

The style is dry and the explanations terse, with comparatively few examples for the mass of information contained in the book – more akin to a reference manual. Correspondingly, the presentation strives for exhaustiveness: every field of every header, every option of every flag, every check by every function is listed and described. There are slip-ups though. Thus, the Unit Control Block (UCB), which is at the core of the VMS I/O subsystem, is repeatedly referred to throughout the book,

and several of its fields appear in a number of relations and validations – but the UCB itself is never described. One must look up its structure in the book "*VMS Internals and Data Structures*".

One becomes rapidly accustomed to DEC conventions – such as calling "word" what is a 16 bit datum in a 32 bit architecture, or the little-endian representation of data. On the other hand, one faces a thicket of mentions of routines and system calls that are obviously relevant to the system programmer who has the complete VMS reference documentation at hand, but only tend to slow down reading for anybody else. For example:

> *The volume allocation lock is initially acquired in protected write mode by the* `MOUNT` *routine* `GET_VOLUME_LOCK` *(in* `CLUSTRMNT`*).*

A few VMS concepts should be looked up beforehand to ease comprehension, but additional background is definitely required for chapter 6, otherwise one gets quickly overwhelmed by the minutiae of context switching and parameter passing discussed therein. Readers are therefore well-advised to have "*Computer Programming and Architecture: the VAX-11*" by Levy and Eckhouse at hand (or some equivalent work) for a refresher on P0 and P1 process space, priority levels, the difference between kernel and executive modes, the VAX process status word, and interrupt handlers. In addition, its section "Input and Output Processing" in chapter 9 is a lucid exposition of the flow of control amongst components handling I/O requests; a very welcome and necessary overview with the right degree of detail to understand what takes place at that level before embarking on chapter 6 of "*VMS File System Internals*".

The complexity of the layer handling low-level I/O arises from a series of optimizations with the aim of running device drivers and associated routines at as low a priority as possible, so as to avoid a monopolization of CPU and channels that would exclude other urgent file requests and possibly miss interrupts. "*Computer Programming and Architecture: the VAX-11*" explains the matter clearly. Experienced OS developers are more able than other software engineers to appreciate all the details about partitioning I/O requests into smaller steps, copying context back and forth, raising flags, switching priority levels, and queuing event handlers.

What will be more familiar to all programmers working in production environments is the overhead due to bookkeeping. VMS is a multi-user OS, and consequently Files-11 must record information about quota allocations for each user, validate limits at each file access, and keep quota utilization up-to-date after each operation – all of which complicate Files-11 data structures and algorithms further.

Numerous forward references make the book more intelligible on a second reading. Thus, windows and the structure of Window Control Blocks are concisely described in section 3.1.3.4, but their role and what "window turns" entail are only explained in section 5.4.9. Similarly, the text makes frequent references to locks and value blocks of various kinds, but all these are discussed systematically only in chapter 7. Initially, the most comprehensible parts are found in chapter 2 (because everything else depends on it) and chapter 8 (because it builds upon everything described previously).

## 4. TYPOS

Published after DEC had passed its apex and before it entered its precipitous decline, the book only saw a revised, and alas untraceable, edition 15 years later. There was indeed a need to improve often awkward sentences and to correct quite a number of regrettable typos. Thus, section 2.3.3.2 states that "*the ident area is usually truncated in extension headers*" – when "*omitted*" is meant. Many more examples of such clumsy expressions can be found throughout the book. Further, in section 6.5.2 a sentence states "*(refer to section 6.5.2)*", while on page 382 the list of functions requiring a File Control Block to be marked stale includes "*deaccessing a file (`DEACCESS`)*" twice. The explanations in table 2-10 mix up the notations for before-image and after-image. Table 2-17 incorrectly states that "*126 bad block entries may be recorded*" – the actual number is 125. As yet a further example, in section 5.4.8.1 we find the following paragraph:

> *If the deaccess function sees that the bad block bit is set in the file header, it sets the `FH2$V_BADBLOCK` bit in the file header. Likewise, `INIT_FCB2`, which initializes the FCB according to the given file header, sets the `FCB$V_BADBLK` bit if the `FH2$V_BADBLOCK` bit is set. Setting `FH2$V_BADBLOCK`, in turn, causes the `DELETE_FILE` routine to send the file to the bad block scanner for deletion.*

It obviously is erroneous, all the more so since `FH2$V_BADBLOCK` does not exist – `FCH$V_BADBLOCK` instead does. The paragraph should be rephrased as follows:

> *If the deaccess function sees that the bad block bit is set in the file control block, it sets the `FCH$V_BADBLOCK` bit in the file header. Likewise, `INIT_FCB2`, which initializes the FCB according to the given file header, sets the `FCB$V_BADBLK` bit if the `FCH$V_BADBLOCK` bit is set. Setting `FCH$V_BADBLOCK`, in turn, causes the `DELETE_FILE` routine to send the file to the bad block scanner for deletion.*

More baffling are the mistakes in some explanatory diagrams. `F11$s<10><10><10>` in figure 7-8 should obviously be `F11$s<10><0><0>` just like in other related diagrams. The running example in section 8.5.1 has been seriously garbled. Pictures 8-13 and 8-14 are all right, but figure 8-16 should take the place of 8-15, followed by what is 8-17, then what is 8-15 (with `BAHT` instead of `BAUT`), then a figure showing `BAHT` and `RUPEE` with mode `CR` granted and `KYAT` with mode `PW` granted, then finally what is 8-18 replacing the completely irrelevant figure 8-19.

Poor editing also caused a number of severe omissions in the description of file headers in chapter 2.

First of all, field `FH2$W_CHECKSUM`, which closes file headers, appears neither in figure 2-2, nor in table 2-2. Second, references to fields `FAT$V_NOSPAN` and `FAT$W_VERSIONS` in section 2.4.1 (explaining the format of directory records) lead nowhere. These are absent from the companion book "*VMS Internals and Data Structures*" too. An earlier internal DEC paper entitled "*Files-11 On-Disk Structure Specification*", and which obviously served as basis for chapter 2 of "*VMS File System Internals*", reveals that these two variables are actually sub-fields of `FH2$W_RECATTR`, along with a number of other crucial attributes defining the organization of the file (sequential, direct, relative, indexed-sequential), its record structure (fixed, variable, variable with a fixed portion, stream), record properties

(length, end of record mark, size of record buckets), and various other file characteristics (such as position of EOF mark or the number of buffers to allocate). The entire chapter 6 of "*Files-11 On-Disk Structure Specification*" should have been included in "*VMS File System Internals*" but is completely missing. The definition of `DIR$B_FLAGS` in directory records is incomplete as well. The aforementioned internal paper lists a possible entry type other than file ids (`DIR$C_FID`): symbolic links (`DIR$C_LINKNAME`) – though at the time the paper was written these were not yet implemented, even if the option bits were already reserved.

Overall, the book looks like a rush job that would have greatly benefited from additional proofreading and substantial editorial re-work.

## 5. COVERAGE

The lack of context makes reading "*VMS File System Internals*" a hard slough. There are topics introductions and overview diagrams – but they look more like reminders for developers already familiar with VMS and eager to plunge into the gory details, than didactic material for outsiders. There are exceptions – notably the section devoted to windows and the chapter on VAXCluster.

Paradoxically, sometimes the book does not provide enough details. A case in point are cache header fields `F11BC$L_POOLAVAIL` (defined as "*number of available buffers in each of the buffer pools*") and `F11BC$W_POOLCNT` (defined as "*count of buffers in each of the buffer pools*"). Anybody instinctively assuming that the number of "available" buffers is a subset of the buffer "count" immediately bumps into the uncomfortable fact that the latter counters are 16 bit variables, while the former are 32 bit variables. There are no explanations about what values those counters are initialized with, when exactly they are incremented, and when they are decremented. There is just a notice that they assume the same value when the file system is quiescent, and another about a very specific circumstance for updating `F11BC$L_POOLAVAIL`. Their distinct roles derive from the fact that buffer "credits" are granted and buffers moved from system-wide pools to process-specific pools prior to launching I/O operations, but the information is insufficient to ascertain their precise semantics.

Consider also the case of the index file header. There is an alternate (or "backup") index file header pointed at by fields containing the block number where to find it. On the other hand, readers must figure out by themselves that the primary index file header is the first one to appear right after the index file bitmap, according to the formula in 2.5.1.7 applied to file ids in table 2-14. As a further example, several checksum fields (such as `HM2$W_CHECKSUM1`) are said to be "*computed by the same sort of algorithm as the header checksum*" – but that algorithm is never elucidated beyond a statement that it is a "*simple additive checksum*"; it is present, as an assembly code snippet, in "*Files-11 On-Disk Structure Specification*".

The text is interspersed with remarks giving the rationale for specific features, but an overarching exposition of the design decisions for Files-11 is missing. One learns why the index file is placed in the middle of the disk by default, or why space allocation is distributed amongst members of a VAXCluster in the way it is, but there

is no discussion of what requirements and constraints led to the final architecture of the file system and its VAXCluster extension. Nor are explanations given about such details as why files are extended, by default, by a number of blocks (i.e. 5) that is not a multiple of the default extent size (i.e. 3 blocks). All that is unsurprising: the book was not conceived as a detailed case study for software engineers in general, but as an exhaustive reference document specially for VMS system programmers.

## 6. EVALUATION

As such, it maintains only a tenuous relevance. VAX computers are museum pieces, while DEC/Alpha and Intel/Itanium systems are inexorably fading away (they also use an updated version of Files-11, i.e. ODS-5). As for the port of VMS to the Intel/X86-64 architecture, it entailed a complete overhaul of the file system, whose new design has little if anything in common with the original Files-11.

The book remains a somewhat interesting addition to the library of people and organizations involved in teaching or in R&D on file management systems, if only for historical reasons. The question is whether it retains its value as a case study for the developer who wants to know “how a file management system works *exactly*”.

The obsolescence of Files-11 is not an appropriate argument contra. VAX/VMS was immensely popular and influential, and one learns by studying such systems. Linear data structures to organize space allocation and in-place update schemes may be superannuated, but it is intriguing to realize that they were designed to handle 2 TiB disks – which must have seemed remarkably future-proof in the late 1970s – and also that one of the possible volume configurations exhibits some faint traits reminiscent of a time when magnetic tapes were used as on-line storage. Handling bad blocks is nowadays taken care of in disk controllers, by proprietary schemes not documented publicly. Files-11 dates from an era when this was an OS task, and the book duly documents how information about bad blocks is recorded on disk and what happens when a device driver encounters a bad block. In this respect, reading about an outdated file system brings some advantages.

The real issue is that the book is laborious to work through, because of its structure, because of the intended readership, and not least because of its many, irritating defects. Attentive perusal and re-reading, flipping back and forth to find the definitions of densely interconnected data structures, and checking supplementary documentation are required to gain an understanding of the matter – a cursory reading is simply pointless. One is left with an appreciation for the complexity of a technically interesting VMS subsystem – and plenty of unanswered questions.

All things considered, I cannot recommend that software engineers looking for a serious, in-depth treatment of a file management system specifically hunt for the original "*VMS File System Internals*”. The later, revised (and confidential) edition about the OpenVMS file system would in principle be a better reference, but appears to have gone out of print soon after its first publication. They should therefore avoid embarking on a frustrating search for a remaining copy and, if absolutely necessary, ought to resort instead to the services of inter-library loans. They may however

consider acquiring either book when stumbling upon a really cheap second-hand copy – and then get hold of the earlier DEC Files-11 specification.

## 7. BIBLIOGRAPHY

[1] Andrew C. Goldenstein: *Files-11 On-Disk Structure Specification*, Digital Equipment Corporation, 1985-01-11. https://web-docs.gsi.de/~kraemer/COLLECTION/VMS/ods2.txt

[2] Lawrence J. Kenah, Ruth E. Goldenberg, Simon F. Bate: *VAX/VMS Internals and Data Structures version 4.4*, Digital Press, 1988, ISBN 1-55558-008-4.

[3] Henry M. Levy, Richard H. Eckhouse: *Computer Programming and Architecture – the VAX-11*, Digital Press, 1980, ISBN 0-932376-07-X.

[4] Kirby McCoy: *VMS File System Internals*, Digital Press, 1990, ISBN 1-55558-056-4.

[5] Brian Schenkenberger: *OpenVMS File System Internals*, Butterworth-Heinemann, 2005, ISBN 978-1555582692.

## REFERENCE

Eduardo Casais: *Retrospective review – VMS file system internals*, technical paper, areppim AG, Köniz, Switzerland, 2021-05-15, 7 pages.



## ABOUT THE AUTHOR

Early in his career, Eduardo Casais programmed DBMS-based MIS software on VAX/VMS. Later he worked on object-oriented software development methods, formal design techniques, Internet service platforms, mobile protocols, and Web development.

## ABOUT AREPPIM AG

areppim AG focuses on the display of quantitative information for the WWW. The site http://stats.areppim.com publishes data on a wide range of topics, presented as intuitive, content-rich charts and often accompanied by concise analyses.

## CONTACT

e-mail: info@areppim.com